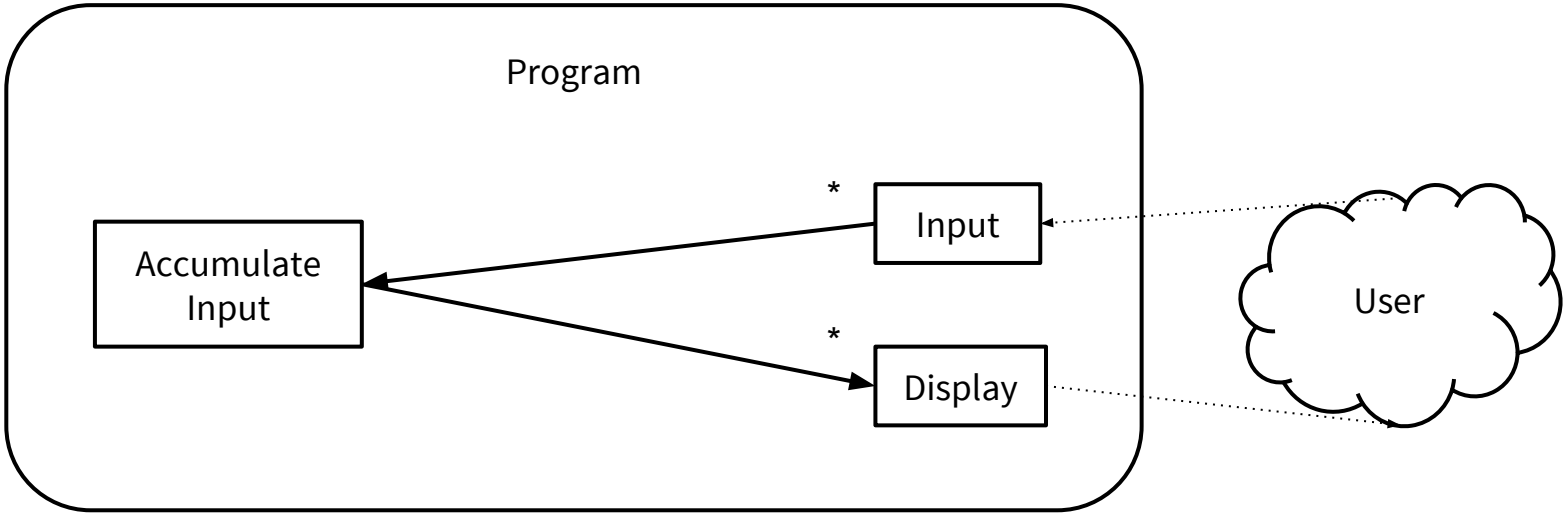
Multi-tier FRP

Combining multi-tier(tierless) languages with functional reactive programming

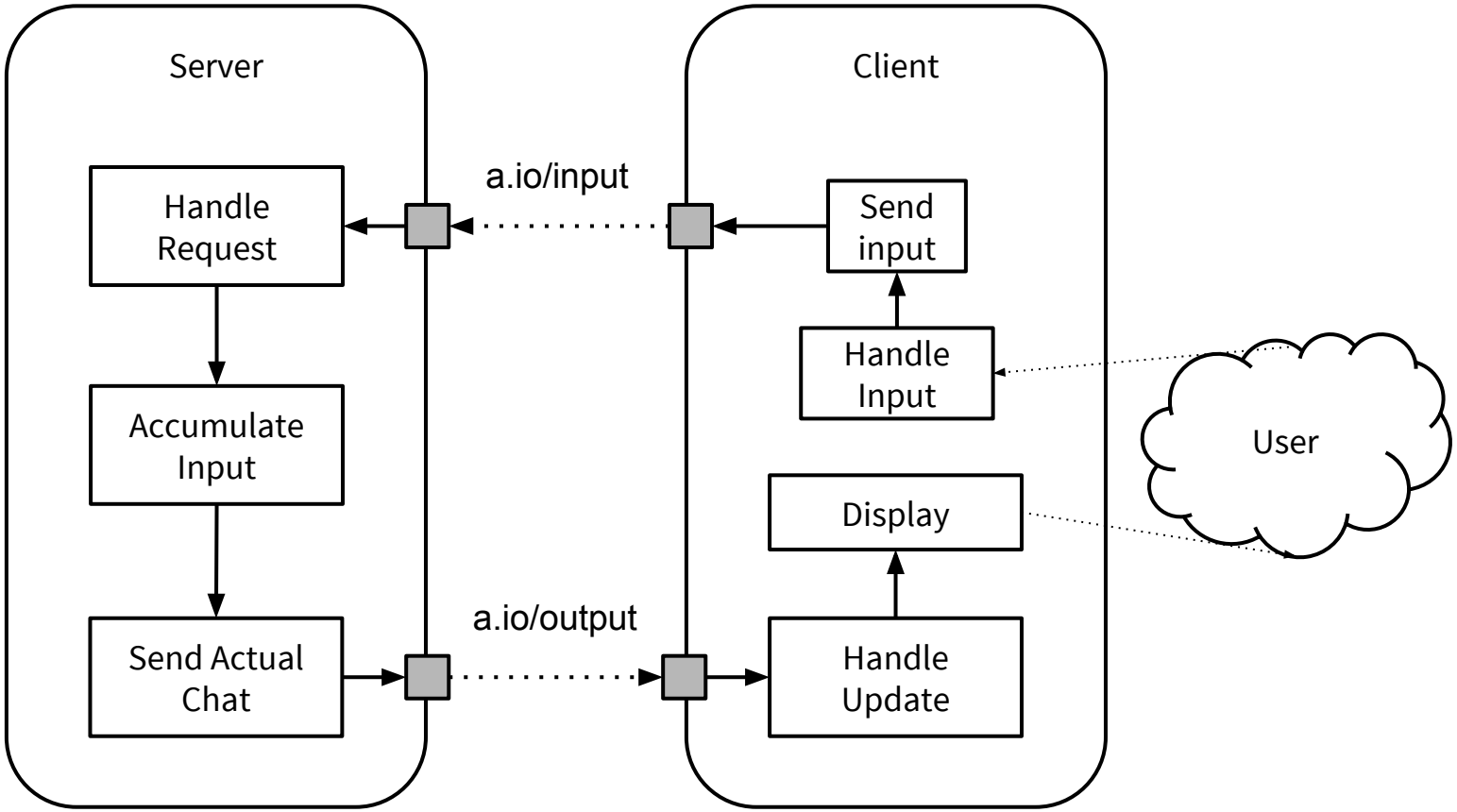
Bob Reynders

Dominique Devriese
Frank Piessens

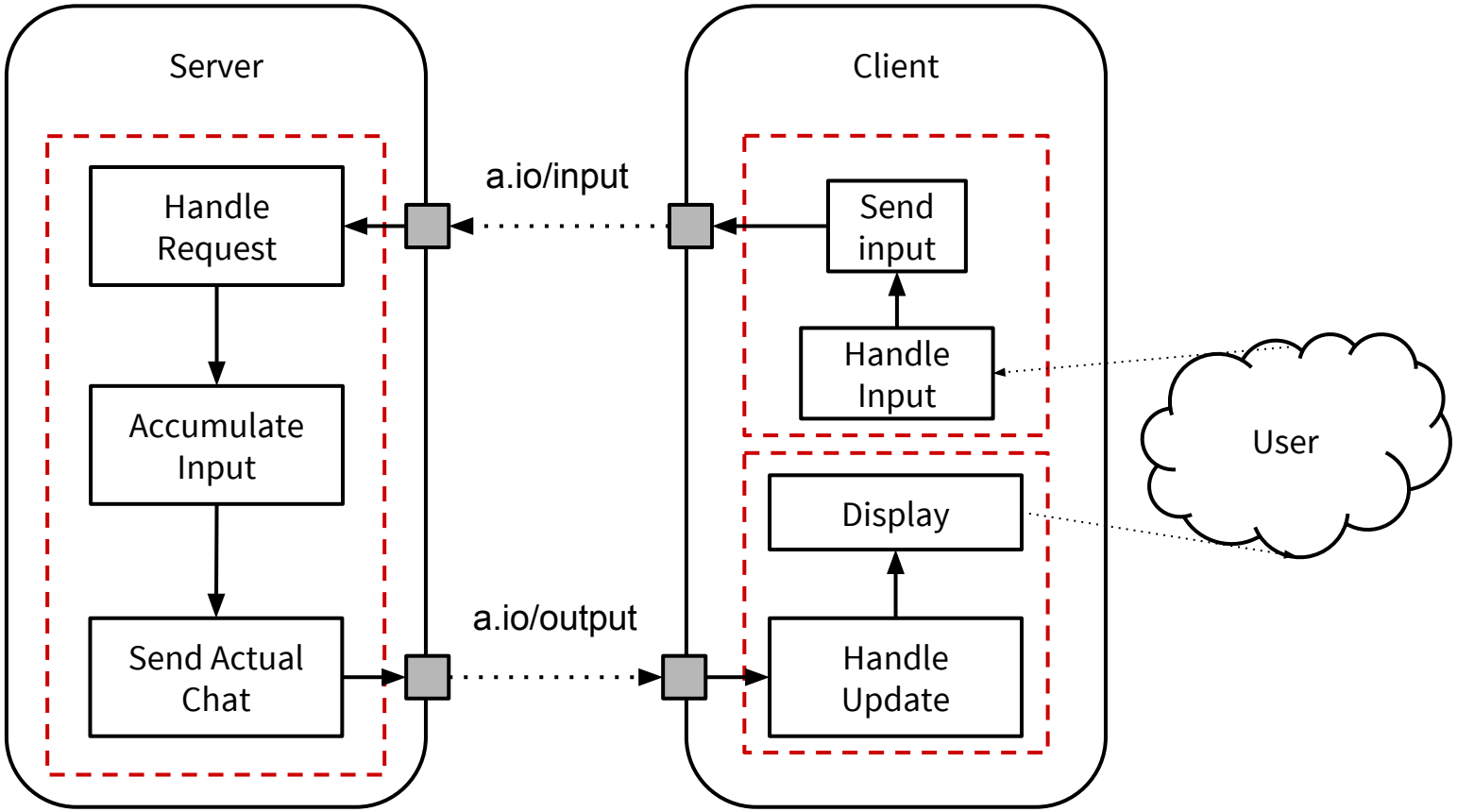
Chat Example
Mental model



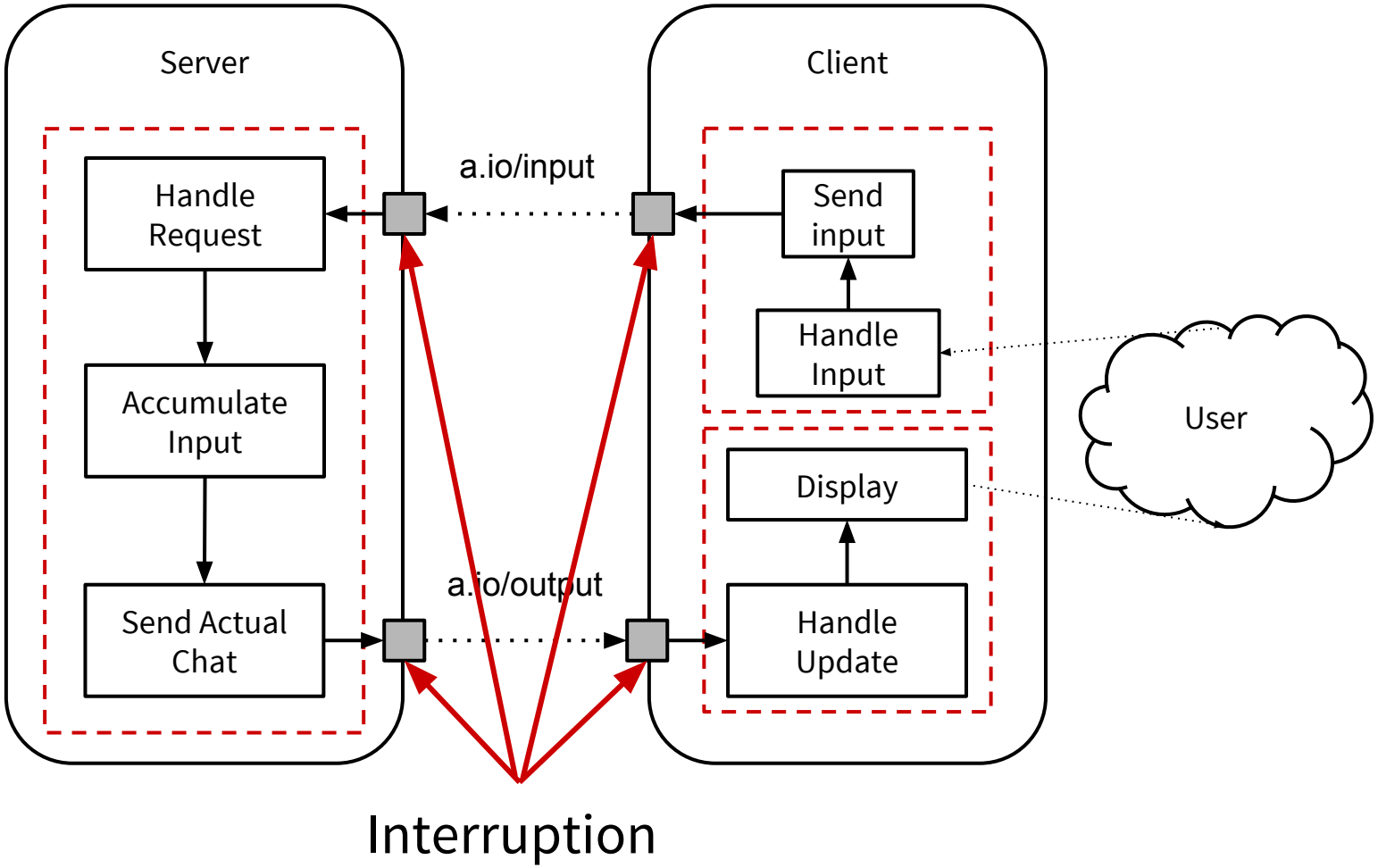
Chat Example
Callback-based



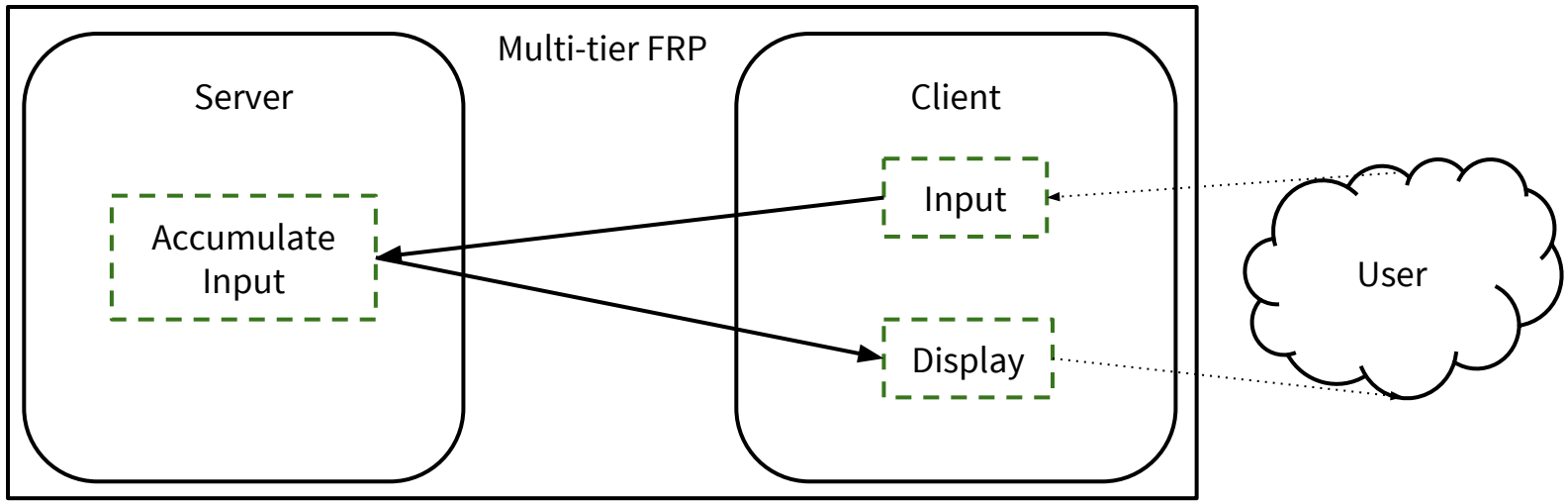
Chat Example
Callback-based Issues



Chat Example
Callback-based Issues



Chat Example
Multi-tier FRP



Functional Reactive Programming

Behavior time-varying value

mouse coordinates: (x, y)

Event discrete value

mouse actions: (action, t0)

Multi-tier FRP API

Classic FRP API

Long-established

```
Event[T].map[A](fun: T => A): Event[A]
Behavior[T].map[A](fun: T => A): Behavior[A]

Event[T].merge[T](e: Event[T]): Event[T]
Behavior[T].combine[A, B](beh: Behavior[A])(fun: (T, A) => B): Behavior[B]

Event[T].hold[T](initial: T): Behavior[T]
Event[T].fold[A](initial: A)(fun: (A, T) => A): Behavior[A]
Behavior[T].sampledBy(e: Event[_]): Event[T]
```

Multi-tier FRP API

One Codebase

Long-established

```
Event[T].map[A](fun: T => A): Event[A]
Behavior[T].map[A](fun: T => A): Behavior[A]

Event[T].merge[T](e: Event[T]): Event[T]
Behavior[T].combine[A, B](beh: Behavior[A])(fun: (T, A) => B): Behavior[B]

Event[T].hold[T](initial: T): Behavior[T]
Event[T].fold[A](initial: A)(fun: (A, T) => A): Behavior[A]
Behavior[T].sampledBy(e: Event[_]): Event[T]
```

```
ServerEvent[T].map[A](f: T => A): ServerEvent[A]
ClientEvent[T].map[A](f: Rep[T] => Rep[A]): ClientEvent[A]
```

Multi-tier FRP API

Explicit Crossing of Tiers

Long-established

```
Event[T].map[A](fun: T => A): Event[A]
Behavior[T].map[A](fun: T => A): Behavior[A]

Event[T].merge[T](e: Event[T]): Event[T]
Behavior[T].combine[A, B](beh: Behavior[A])(fun: (T, A) => B): Behavior[B]

Event[T].hold[T](initial: T): Behavior[T]
Event[T].fold[A](initial: A)(fun: (A, T) => A): Behavior[A]
Behavior[T].sampledBy(e: Event[_]): Event[T]
```

```
ServerEvent[T].map[A](f: T => A): ServerEvent[A]
ClientEvent[T].map[A](f: Rep[T] => Rep[A]): ClientEvent[A]
```

```
ClientEvent[T].toServerAnon: ServerEvent[T]
ServerEvent[T].toAllClients: ClientEvent[T]
ServerBehavior[T].toAllClients: ClientBehavior[T]
```

Implementation

JavaScript

BaconJS

Scala

ScalaReactive Core

JS-Scala & LMS

Spray

Multi-tier Functional Reactive Programming for the Web

Bob Reynders, Dominique Devriese, Frank Piessens

SPLASH Onward! 2014, Accepted

Multi-tier Functional Reactive Programming for the Web

Bob Reynders
iMinds - Distrinet, KU Leuven
bob.reynders@student.kuleuven.be

Dominique Devriese Frank Piessens
iMinds - Distrinet, KU Leuven
{firstname.lastname}@cs.kuleuven.be

Abstract

The development of robust and efficient interactive web applications is challenging, because developers have to deal with multiple programming languages, asynchronous events, propagating data and events between clients and servers, data consistency and much more. Several approaches for (partly) addressing these challenges have been proposed. Two relevant ones are (1) multi-tier languages and (2) functional reactive programming (FRP). Multi-tier programming languages support the development of client and server in a single language, and hide much of the complexity related to distribution. FRP offers the right abstractions to make event-driven programming convenient, safe and composable. However, existing web frameworks and programming languages exploit the benefits of both approaches separately, for example by restricting the use of FRP to the client side.

We propose *multi-tier FRP for the Web*, a novel approach to writing web applications that deeply integrates FRP and multi-tier languages, and where the whole is greater than the sum of its parts. In multi-tier FRP, the developer programs server and client together as an FRP application composed of behaviors (signals) and events. He/she chooses explicitly where the boundary between server and client is crossed. To make our approach more concrete and provide evidence of its potential, this paper presents a concrete design and implementation of a multi-tier FRP API for the web in the programming language Scala, using an embedded JavaScript DSL that makes Scala usable as a multi-tier language. This allows us to present initial evidence of the benefits of the multi-tier FRP approach on example applications, and to experiment with possible answers to the remaining questions. Concretely, we show possible solutions for problems like exposing client identity on the server and efficiently preloading clients with the latest application state. Our results show that multi-tier FRP is a promising, declarative, yet practical way of writing web applications.

http://dx.doi.org/10.1145/

Keywords Functional Reactive Programming, FRP, multi-tier web framework.

1. Introduction

Developing interactive web applications presents a number of interesting challenges for programmers. One important challenge is the inherent distributed nature of the platform with parts of the application running on the server and other parts on the (zero or more) clients. Another challenge is dealing with the asynchronous communication that is inherent to user communication and typically used in the web's client-server communication for reasons of performance, failure tolerance and responsiveness towards the user.

The standard approaches for dealing with these challenges (the use of callbacks and separate server- and client-side codebases, typically in different programming languages) present significant downsides. In recent years, interest in web application development has not ceased to increase (both in research and industry) and several novel approaches have been proposed to improve over these approaches. In this paper, we focus on two such novel solutions specifically: Functional Reactive Programming (FRP) and multi-tier languages.

Asynchronous communication and FRP The standard approach for dealing with asynchronous user and client-server input and output is the use of *callbacks*: imperative components that are invoked in response to asynchronous events. Specifically, web applications use JavaScript event handlers on the client side and HTTP request handlers on the server side. A web application containing many such callbacks, all potentially modifying the application's mutable state, may have a very complex control flow within and across both parts of the application. Such code can be very difficult to reason about.

FRP (Elliott and Hudak 1997) (although initially proposed for modelling animations) can be used as an alternative programming model for asynchronous applications. Instead of using side-effecting callbacks, the program is constructed by composing *behaviors* (also known as *signals*) and *events*: components representing time-dependent values.

github.com/tzbob/s-mt-frp
bob.reynders@student.kuleuven.be